Université de Montpellier - Master 1
Informatique

Data Science

Classification de document de Fake news

Auteur :
Elbachir REHHALI : 21812980
Mohamed HASSAN IBRAHIM : 21914461
Karim DAHDOUH : 21914506
Thierno BARRY : 21914683

14 mai 2020

Table des matières

Chapitre 1

Introduction

Aujourd'hui, on trouve sur Internet et dans les journaux un grand nombre de données quantitatives, mais aussi et surtout, des données textuelles qualitatives provenant de figures politiques. Bon nombres de sites web offrent aussi la possibilité aux clients de déposer leur articles en ligne avec le nom de l'auteur au soit anonymiser et Très vite la véracité de cet article est alors recherché d'où l'intérêt du Fact-checking.
Le terme anglais fact-checking, littéralement « vérification des faits », désigne un mode de traitement journalistique, consistant à vérifier de manière systématique des affirmations de responsables politiques ou des éléments du débat public.

Comme l'explique Dominique Cardon dans son livre La Démocratie Internet, la pertinence attribuée à une information ne résulte plus d'une évaluation normative de son contenu par des experts mais émane plutôt d'une « agrégation numérique ». C'est-à-dire que, désormais, les informations exposées sur la toile ne sont plus filtrées en étant au préalable passées au crible par des experts et journalistes. Elles sont à la place hiérarchisées a posteriori par des algorithmes de classification qui les trient.
Du coup L'objectif de ce projet est de créer une intelligence artificielle (classifieur) qui permettrait, à partir d'un article, de reconnaître la polarité de celle-ci : est-ce que l'article est vrai ou faux ou bien plus encore un mélange de ces 2 valeurs (Mixture) ? Comment lui apprendre à reconnaître la négation ou bien encore plus complexe, le sarcasme ? Nous essaierons d'y répondre par la suite.

1.1 Context

Ce projet de science des données consiste à faire du fact-checking de manière automatisé ou encore la classification d'assertions selon leur valeurs de véracité. Nous avons donc un jeu de données collecté à partir de sites spécialisés dans ce domaine comme (politifact, snopes, etc). En effet, nous devons à partir de ces données déjà recueillies, faire de la classification. Nous allons nous focaliser sur deux tâches de classification à savoir : 1. Vrai vs Faux. 2. Vrai et Faux vs Mixture. Dans les deux cas, la classification est binaire.

Chapitre 2

Features Engineering

2.1 Choix des features

Avant de faire une quelconque prédiction, il faut d'abord sélectionner les variables d'intérêts encore appelées features . Nous avons des données brutes avec plusieurs colonnes dont beaucoup ne représentant aucune utilité pour notre travail comme l'id. Nous avons fait une petite liste des colonnes de départ figurant dans le jeu de données dont la principale feature à savoir le (texte) et les meta-data qui l'accompagnent : (id, text, date, ratingName, author, headline, named entities claim, named entities article, keywords, source, sourceURL, link, langage, truthRating). Au total quatorze(14) features possibles. Cependant, c'est à nous de voir quelles sont les variables d'intérêt dans cette liste. Pour commencer, nous pouvons déjà effectuer une première suppression concernant les colonnes qui ne contiennent aucune utilité pour la classification. Nous pouvons donc citer entre autre :

— id
— link
— langage (inutile, toutes les assertions sont en anglais)
— sourceURL

ces variables ne nous sont d'aucune utilité. l'id permet juste d'identifier une assertion. Le link et sourceURL font référence à la source de l'article. Nous avons donc décidé de garder la source. Nous nous sommes ensuite intéressé aux variables qui pourront nous être utiles dans le processus de classification. Par ailleurs, nous avons donc choisi les potentiels meta-data, qui peuvent s'avérer pertinent. Notre principale feature est le text. Les potentielles features qu'on a sélectionnées sont :

— text
— author
— headline
— source
— keywords

(A Noter que ces choix des features ci-dessus ont été fait minutieusement dans le notebook en effectuants plusieurs classification bien avant et en adoptant une démarche exploratoire et itérative).

Nous avons donc ajouté quatre (4) autres variables d'apprentissage en plus du texte.

En effet, tout le monde est unanime sur le fait que face à une information, la première des chose à vérifier sur cette dernière c'est sa source. En l'absence de source, une information perd toute sa fiabilité. Comme dans notre contexte ce sont des assertions de figures politiques que nous traitons,

l'auteur peut donc s'avérer pertinent. ci- dessous un graphe qui donne l'exemple de véracité des assertions de Donald Trump.

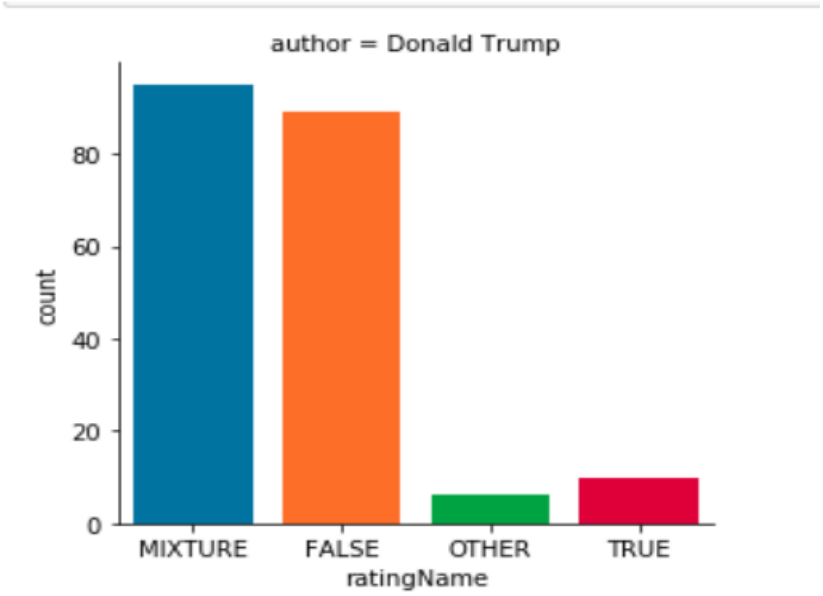

FIGURE 2.1 – Graphe des taux de veracité pour Donald Trump

2.2 Pré-traitement des données

Avant de nous lancer directement dans le vif du sujet c'est à dire la classifications des assertions selon leur valeur de véracité, nous devons bien entendu avoir déjà fait la partie "pré-traitement des données" qui est une étape indispensable avant la construction notre modèle de classification.
Du coup, pour bien nettoyer notre jeu de données, nous avons effectué plusieurs transformations suivantes :

- **Suppression des données manquantes**
- **Traitement des données déséquilibrées (OverSampling)**
- **Vectorisation (TF-IDF)**
- **Transformations sur des features**

2.2.1 Suppression des données manquantes

Dans cette partie on s'intéresse aux valeurs vide dans notre dataset, il existe deux choix pour gérer ces valeurs :

- **Supprimer la ligne contenant la valeur vide :** Il s'agit de supprimer toutes des lignes pour lesquelles on trouve au moins un élément manquant. Car les valeurs null peuvent influer négativement les résultats de classification des assertions.
- **Remplacer les valeurs vides :** Dans ce cas on remplace les valeurs vides par une chaîne de caractères qu'on peut le considérer comme une valeurs qui influence l'apprentissage, comme par exemple le cas dans author on trouve la valeur "unknown" qui signifie qu'on a pas de valeur.

Si on considère les features suivantes : “text, headline, source et truthRating”. Comme il est illustré dans la figure ci-dessous, on remarque la présence des valeurs manquantes au niveau de feature “headline” . Donc, un total de 118 lignes sera supprimées de notre dataset.

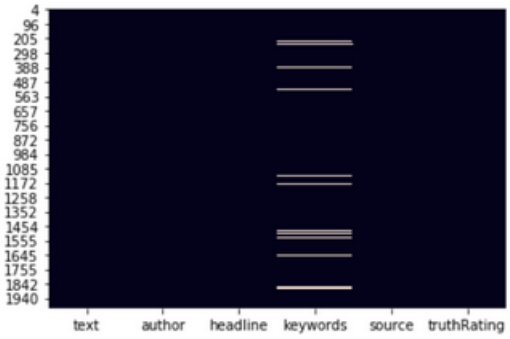

FIGURE 2.2 – Graphe des valeurs manquantes

2.2.2 Traitement des données déséquilibrées (OverSampling)

Afin d’équilibrer le jeu de données en classes homogènes, il existe, généralement, deux approches. UnderSampling consiste à enlever des données de la classe majoritaires, des assertions ayant comme valeurs de truth Rating 1 et 2. c’est la solution la plus simple mais pas la plus efficace. La deuxième approche est OverSampling qui permet d’ajouter des données aux classes minoritaires (-1 et 0). EN effet, c’est la meilleure solution dans notre cas.

Pour avoir une vision sur notre dataset, nous avons essayé d’afficher les statistiques simples sur la distribution des variables à prédire dans le jeu de données des assertions. La figure suivantes montre très clairement la répartition les différentes classes de truth Rating.

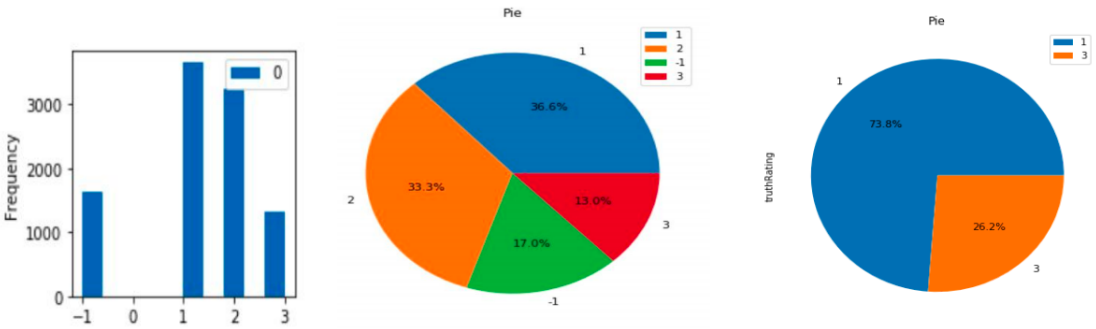

FIGURE 2.3 – Histogramme des valeurs de truth Rating

Dans le cas de du premier dataset (True vs False), nous voyons bien à partir de la visualisation de la figure ci-dessus qu’il y a une grande différence entre les classes TRUE & FALSE . En fait, nous allons rééchantillonner au hasard l’ensemble de données d’apprentissage et ainsi dupliquer des exemples de la classe minoritaire, concept appelés suréchantillonnage. Après avoir appliquer la technique d’oversampling nous obtenons un jeu de données équilibré avec même nombre d’assertions True et False.

2.2.3 Vectorisation (TF-IDF)

La bibliothèque NLTK (tokenisation) à été mise en place pour transformer nos données textuelles en séquences de mots permettant de découper une phrase en tokens à l'aide de "word tokenize". Une fois que les features ont été transformé en une séquence de mot, nous avons appliqué la technique de vectorisation TF-IDF. Cette dernière étant plus performante que celle de CountVectorizer car elle prend en considération la longueur du corpus, ainsi le nombre d'occurrence de chaque mot. En se basant, TF-IDF attribue un poids aux tokens. Par contre CountVectorizer compte juste la fréquence du mot dans le texte quelque soit sa taille.

2.2.4 Transformations sur des features :

Dans cette partie on s'est concentrer sur les différents features pour ainsi faire différentes optimisations possibles :

1. **Suppression des caracteres non-ASCII :** Dans notre cas les caractéres non-ASCII n'ont pas vraiment de poids, du coup on a opté pour une suppression car ce sont des caractères qui ne sont pas lourds de sens et qui sont souvent ecrit par erreur ou pour interpretation diverse.
   Dans notre fonction **remove_non_ascii** on a essayer de faire appelle à trois sous-fonctions :
   — **unicodedata.normalize('NFKD', word)** pour la normalisation Unicode.
   — **encode('ascii', 'ignore')** pour encoder les caractères normaliser en ASCII en ignorant les caractères non ASCII.
   — **decode('utf-8', 'ignore')** un décodage en UTF-8 des octets en ASCII prenant compte les caractères en ASCII.
2. **Convertir les variables d'apprentissage en minuscule ou Lowering Case :** Les majuscules sont souvent utiliser au début de la phrase ou soit pour déclarer les noms propres, dans notre cas on a pas besoins de faire la différence entre les alphabets en majuscule et en minuscule du coup mettre tout les mots en minuscule est un choix judicieux .
   Ainsi 1 meme mot quelque soit minuscule ou en majuscule sera interpreté identiquement .
3. **Supprimer les ponctuation et les caractères spéciale :** Nous avons pensés à supprimer les ponctuations et les caractères spéciaux dans le but de réduire le nombres des mots différents puisqu'on a pas besoins de ces caractères.
4. **Remplacer les nombres en lettres.**
5. **Substituer plusieurs espaces par un seul**
6. **Supprimer les stopwords** Pour le traitement des stopwords, nous avons considéré le dictionnaire des stopwords offert par NLTK pour le langue anglaise. Dans ces stopwords on a vu que le "not" se supprime alors on a ajouter l'exception pour ne pas considérer comme un stopwords.
7. **Lemmatisation du texte** La lemmatisation est le processus de regroupement des différentes formes fléchies d'un mot afin qu'elles puissent être analysées comme un seul élément. La lemmatisation est similaire à la racine mais elle apporte du contexte aux mots. Il relie donc des mots ayant une signification similaire à un mot. C'est un pré-traitement utile et important pour rassembler les mots ayant le même sens.
   Dans notre cas on a essayer d'utiliser une méthode Lemmatize(word).
8. **Prendre en compte la suite de deux ou plusieurs mots consécutifs (ngrams)**
9. **Ajouter une correction orthographique dans le cas ou certains mots serait mal orthographié et ainsi mal interprété .**

2.3 Mise en place d'une pipeline

Après avoir choisi le modèle , nous créons un pipeline permettant d'enchaîner les étapes de pré-traitement, la vectorisation et l'apprentissage d'un modèle. Il va permettre d'appliquer à une séquence de données (features) d'être transformées et corrélées ensemble. Ensuite, pour associer les différentes transformations en un seul pipeline, nous avons utilisé la classe FeatureUnion permettant ainsi de concaténer alors les résultats des chacunes des transformateurs. Ceci est utile pour combiner plusieurs mécanismes d'extraction de fonctionnalités en un seul transformateur.

2.4 Train Test du Dataset

Il est important de diviser le jeu des données préparées en une base des données d'apprentissage (70 %) et une autre pour le test (30%) pour évaluer les performances du notre système. Les données d'apprentissage sont utilisées pour construire le modèle de classification, tandis que la base des données de test vise à évaluer notre classifieur.

Chapitre 3

Résultats et analyse de classification

Maintenant que le choix des features est fait et que nos données sont bien préparées, la seconde étape la plus cruciale peut commencer c'est à dire l'emploi de nos features pour entraîner nos modèles de classification. Cependant les méthodes de classifications etant plusieurs et variées la question qu'il nous faut nous poser à ce moment du projet est donc : Quel serait l'algorithme de classification le plus adapté et qui ainsi pourrait nous donner le meilleur rendu (accuracy, precision, recall, etc) possible ? Ci-dessus liste les principaux algorithme de classification qu'on utilisés :

1. Arbre de décision (Decision Tree)
2. Naive Bayes (GaussianNB)
3. KNeighborsClassifier
4. Linear SVC
5. RandomForest

Par contre on sait d'avance que Gaussian Naive bayes s'avère très efficace pour la classification avec des données textuelles.

3.1 Cross validation

Une fois la vectorisation effectuée, nous préparons un ensemble de modèles à tester sur l'ensemble des features choisies. Pour assurer la bonne performance des classifieurs, nous utilisons une cross-validation sur 10 partitions différentes du dataset et la métrique Accuracy pour évaluer leurs performances. Pour que notre modèle soit appris sur plusieurs jeux de données, on a appliqué 10-fold cross validation pour évaluer la qualité du modèle. En effet, Le jeu de données sera découpé en 10 partie. Du coup, notre modèle sera entraîné sur neuf (9) parties et testé sur une. Pour chaque modèle, nous calculons le score pour chaque partition, puis la valeur moyenne et la déviation standard de l'ensemble des scores. Pour le faire nous utilisons KFold de la bibliothèque scikit-learn.

3.1.1 Résultats de la cross-validation Dataset {TRUE} vs. {FALSE}

Modèle	Score moyen	Déviation standard
RandomForest	93.3%	2.22%
GaussianNB	90,46%	2.53%
LinearSVC	90,17%	1.85%
DecisionTree	87.48%	3.63%
KNN	73,5%	3.9%
SVM	46.29%	2.13%

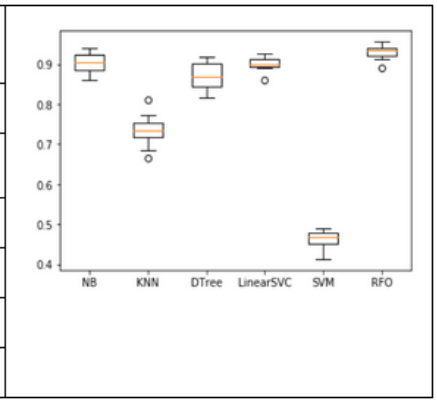

FIGURE 3.1 – Résultats de la cross validation {TRUE} vs. {FALSE}

A partir des résultats des différents classifieurs appliqués aux deux datasets, comme l'illustre la Figures 3.1 , nous remarquons que ceux-ci sont très proches. En appliquant la cross-validation sur l'ensemble des modèles choisis, nous nous retrouvons avec les résultats de la Figure 3.1 ordonnés par ordre décroissant sur les scores moyens. En s'appuyant sur ces résultats, nous choisissons ainsi les modèles RandomForest (93.3%) et Gaussian Naives bayes (90.46%, 2.53%) pour la suite. Cependant le modèle Gaussian Naives bayes etant un peu plus performant que celui de Random Forest car ce dernier depend un peu du hasard.

3.1.2 Résultats de la cross-validation Dataset {TRUE et FALSE} vs. {MIXTURE}

Modèle	Accuracy moyen	Déviation standard
Random Forest	76.43%	38.61%
LinearSVC	76.86%	31.77%
GaussianNB	74.32%	2.96%
Decision Tree	72.81%	3.36%
KNN	67.83%	3.78%
SVM	47.72%	20.23%

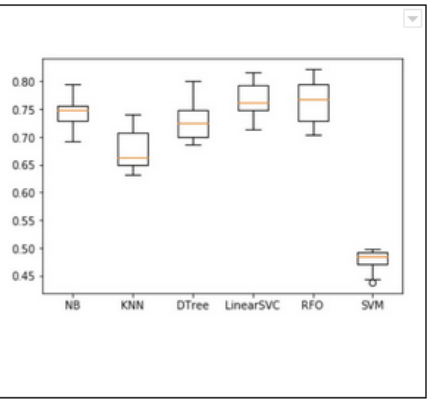

FIGURE 3.2 – Résultats de la cross validation {TRUE et FALSE} vs. {MIXTURE}

Pour la deuxième classification True et False vs sMixture, en se basant sur les résultats de l'accuracy moyenne, le Random forest est toujours le meilleur classifieur avec une valeur d'accury de 76.43% mais Naives bayes n'est egalement pas loin avec une bonne valeur d'accuracy (76.32%).

3.2 Matrice de confusion True vs False

Afin de mesurer les performances de nos modèles de classification, nous nous sommes basés sur la matrice de confusion (Confusion Matrix) qui permet de vérifier notamment à quelle fréquence les prédictions sont exactes par rapport à la réalité. Nous sommes intéressés essentiellement à l'analyse de la matrice de confusion des classifieurs Random Forest et GausianNB.

Pour la seconde partie de classification TRUE et False vs Mixture , on a besoin de modifier notre dataset afin de répondre convenablement à la problématique, nous avons unifié les valeurs des TruthRathing vrai et faux face a mixture ainsi on aura une classification binarie.

3.2.1 Classification Naive Bayes

	precision	recall	f1-score	support
FALSE (1)	0.90	0.80	0.85	197
TRUE (3)	0.83	0.92	0.87	216
accuracy			0.86	413
macro avg	0.87	0.86	0.86	413
weighted avg	0.87	0.86	0.86	413

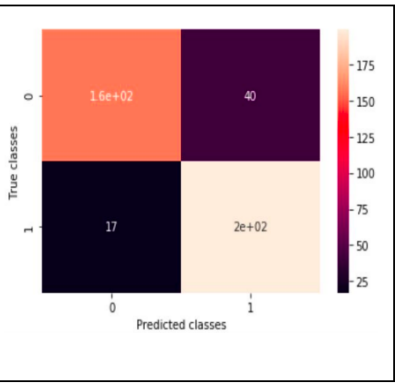

FIGURE 3.3 – Matrice de confusion de GausianNB (Vrai vs Faux)

Le résultats de la classification Naive Bayes montre que La plupart des prédictions sont sur la ligne diagonale de la matrice de confusion (vrais positifs) et (faux négatifs) comme le illustré dans la Figure 3.3.

3.2.2 Classification Random Forest

	precision	recall	f1-score	support
FALSE (1)	0.89	0.92	0.90	197
TRUE (3)	0.92	0.90	0.91	216
accuracy			0.91	413
macro avg	0.91	0.91	0.91	413
weighted avg	0.91	0.91	0.91	413

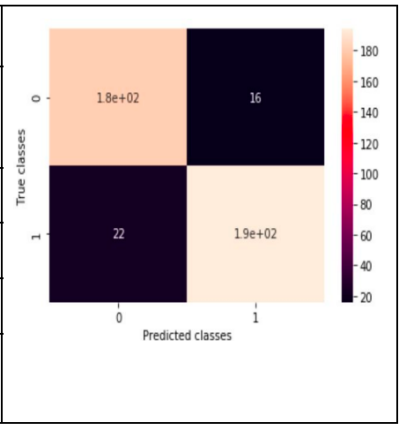

FIGURE 3.4 – Matrice de confusion de GausianNB (Vrai vs Faux)

À partir de ces résultats, on constate qu'on a obtenu des excellentes mesures de précision (89% et 92%), de recall(>90%) et de f1-score(90%), ce qui démontre la qualité des assertions classifiées par nos modèle de classification.

3.3 Mise en place d'un Gridsearch et Hyper-parameters

Grid search est un processus qui consiste à effectuer un réglage hyperparamétrique afin de déterminer les valeurs optimales pour un modèle donné. Ceci est important car les performances de l'ensemble du modèle sont basées sur les valeurs d'hyperparamètre spécifiées.

Après la phase de cross-validation, il nous faut trouver les meilleurs hyperparamètres permettant de raffiner les régions de décision de chaque modèle choisi afin d'avoir les meilleurs prédictions possibles (RandomForest , Decision Tree etc) . On applique la méthode cross validation avec GridSearchCV de la bibliothèque scikit-learn (elle effectuera une recherche de grille et va permettre de tester les différentes combinaisons de valeur des hyperparamètres fournis pour chaque modèle).

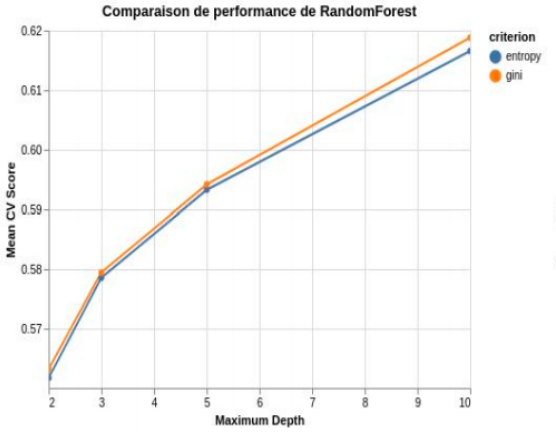

FIGURE 3.5 – GridSearch du Random Forest

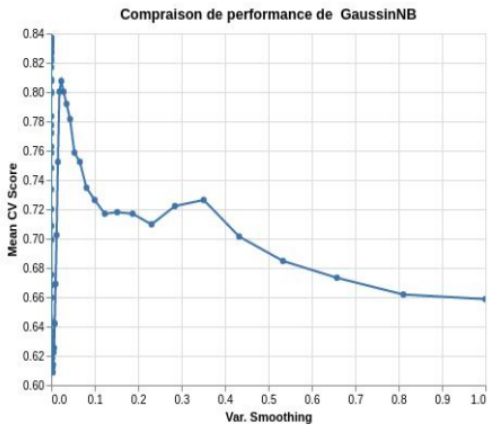

FIGURE 3.6 – GridSearch du GaussainNB

À partir de l'histogramme, nous pouvons remarquer que les meilleurs hyperparamètres sont les suivants : entropy avec une profondeur maximale de 10 et une valeur min_samples_split de 2 pour le modele RandomForest. Après on a appliqué GridSearch sur les trois modèle, à savroir : GaussainNB, Random Forest et Decission, les résultats démontrent que Naive Bayes donne le meilleur score d'accuracy pour ces hyperparamètres sur jeu de test(76.7%).

Pour le deuxième dataset True et False vs Mixture, on a obtenu les résultats suivant :

Classifieur	Meilleurs Paramètres	Meilleur score	Meilleur estimateur
Random Forest	{'criterion': 'entropy', 'max_depth': 10, 'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 9}	71.48%	(ccp_alpha=0.0, class_weight=None, criterion='entropy', max_depth=10, max_features='auto',....)
Gaussian Naive bayes	{'var_smoothing': 1.232846739442066e-06}	83.66%	GaussianNB(priors=None, var_smoothing=1.232846739442066e-06)

Classifieur	Meilleurs Paramètres	Meilleur score	Meilleur estimateur
Random Forest	{'criterion': 'entropy', 'max_depth': 10, 'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 9}	71.48%	(ccp_alpha=0.0, class_weight=None, criterion='entropy', max_depth=10, max_features='auto',....)
Gaussian Naive bayes	{'var_smoothing': 1.232846739442066e-06}	83.66%	GaussianNB(priors=None, var_smoothing=1.232846739442066e-06)

En analysant les résultats du tableau ci-dessus, nous pouvons constater que le classifieur Naive Bayes est le meilleur. et c'est aussi ce résultat que nous a donné l'algorithme sur la fonction Grid Search.

3.4 Sauvegarde de pipeline

La dernière étape consiste à la sauvegarde du pipeline du classifieur GaussainNB avec les prétraitements associées . Cela va nous permettre de le charger et de l'utiliser facilement par la suite . Le tableau suivant présente les résultats de prédiction de 15 assertions de notre jeu de données True vs False.

	réelles	prédites
assertion : 4	réelle 1	prédite 1
assertion : 7	réelle 1	prédite 1
assertion : 9	réelle 1	prédite 1
assertion : 10	réelle 3	prédite 1
assertion : 11	réelle 1	prédite 1
assertion : 19	réelle 1	prédite 1
assertion : 21	réelle 1	prédite 1
assertion : 22	réelle 1	prédite 1
assertion : 24	réelle 1	prédite 1
assertion : 25	réelle 1	prédite 1
assertion : 26	réelle 1	prédite 1
assertion : 27	réelle 1	prédite 1
assertion : 28	réelle 1	prédite 1
assertion : 30	réelle 1	prédite 1
assertion : 31	réelle 3	prédite 3

FIGURE 3.7 – Resultat prédiction sur 15 assertions

Chapitre 4

Conclusion

Toutes les évocations actuelles du fact-checking sont fondées sur l'observation de pratiques journalistiques relativement récentes, qui s'étendent un peu partout dans le monde, et qui consistent à vérifier la véracité de propos tenus par des responsables politiques ou d'autres personnalités publiques. Ce domaine étant en pleine évolution avec diverses applications. Au cours de ces dernières années, il y a eu des progrès remarquables afin de répondre à une forte demande des sociétés cherchant à catégoriser la véracité de leurs articles Cependant, cette tâche paraît assez limitée en adoptant une approche machine learning pure, et nécessite des approfondissements afin d'effectuer plus d'analyses contextuelles et sémantiques en améliorant la sélection des features lors de la vectorisation.

Pour conclure, ce projet a été une véritable opportunité pour appliquer ce que nous avions vu en cours et en travaux pratiques sur un cas réel. Nous avons appliqué plusieurs techniques de data science. En effet, ce projet nous a permis : de suivre une méthodologie de travail bien étudié, d'approfondir nos connaissances dans l'ingénierie de données et, et enfin de travailler avec différents modèles de classification pour classifier les assertions. Nous en avons tiré un grand bénéfice, aussi bien au niveau technique qu'au niveau personnel. La réalisation de ce travail a été d'une importance très considérable.